INTERNATIONAL JOURNAL OF ELECTRICAL ENGINEERING & TECHNOLOGY (IJEET)

ISSN 0976 – 6545(Print)
ISSN 0976 – 6553(Online)
Volume 6, Issue 3, March (2015), pp. 07-16
© IAEME: www.iaeme.com/IJEET.asp
Journal Impact Factor (2015): 7.7385 (Calculated by GISI)
www.jifactor.com

IJEET

©IAEME

A COMPUTATION MODEL FOR REAL-TIME SYSTEMS

NDZANA Benoît

Senior Lecturer, National Advanced School of Engineering,
University of Yaounde I, Cameroon

BIYA MOTTO

Frederic, Senior Lecturer, Faculty of Sciences,
University of Yaounde I, Cameroon

LEKINI NKODO Claude Bernard

P.H.D. Student; National Advanced School of Engineering,
University of Yaounde I, Cameroon

ABSTRACT

This paper examines a simplified computation model for real-time systems, TMRS (Transition Model for Real-time Systems). It investigates a simpler way to develop a model in order to highlight the main aspects and properties of it and mainly timing constraints problem. Such model takes into account the physical process and the controlling process (computer process); the physical process being either a discrete-event system such as a manufacturing process, or continuous system such as a chemical process. The controlling process is an active program being executed to control and monitor the physical process; the controlling process is often denoted by reactive systems. The whole environment is considered in this study. The difference of this model with some proposed models is discussed. Such model enables to discuss properties of real-time systems and mainly timing constraints as a safety property using a temporal logic based specification language are based on temporal logic and transition systems; the communication model between computation processes and physical processes is based on CSP formalism. The contents of the paper will be limited to the approach, the definition of the model and how to specify its invariant properties.

Keywords: Automatic Control, Computation Model, Proof System, Real-Time Systems, Temporal Logic, Temporal Semantics, Timing Constraint, Transition Systems.

INTRODUCTION

Advances in the theory of programming and computation over the last three decades have enabled the introduction of an increasing amount of mathematical formality. The application of these formal methods leads to systems that are developed by means of mathematical and engineering disciplines. A formal method of system development can be decomposed in three main or major components as follows:

- Formal notation for describing specifications and designs in a mathematical precise manner.
- A collection of inference/proof system for demonstrating that implementations meet their specification. The specification meant here is based on the definition developed by Lamport [1].
- A methodological framework for deriving implementations from specifications.

Our concern is directed towards reactive systems with a particular emphasis on temporal properties: timing constraints. The paper is structured as follows: the first section is an introduction on real-time systems and the necessity to develop a computation model. The most fundamental concepts will be illustrated through some existing models. The use of temporal logic for defining temporal semantics will be outlined. The second models introduces a complexity in evaluating the timing constraints problem. Our approach reduces the indeterminism by taking into account the physical process dynamics (temporal aspects) into the model. The basis of the approach is to give a formal solution with a pragmatic view of the problem. Temporal logic is introduced in the third section. In the fourth section, the computation model will be defined. A specification on safety assertions based on temporal logic will be presented. The safety properties specification and mainly timing constraints will be discussed in the fifth section. An attempt to define formally the interaction between both processes will be done.

I. REAL-TIME SYSTEMS

Programming has been classified into three main categories [2]: sequential programming, parallel (multi) programming and real-time programming. Real-time programming is carried either some classical languages or specific ones called “real-time languages” [3].

Automatic control discipline is the basis of linking a c ontroller (being actually implemented on a computer system) and a physical process. Control scientists were concerned first on analysis study; the main property being the stability property. Some verification and correction methods exist to guarantee the stability using linear algebra (for linear systems) [4]. The step to control a system is a systematic way of development of control and automation. The control/supervisor is now implemented on a computer system. The discipline in programming a control algorithm, real-time programming, has emerged since: the aim is to make control software as time efficient as it can be made entirely through hardware.

Real-time programming has been defined and introduced as a discipline for its specific requirements with respect to the timing efficiency. Hence, the main difference with other categories mentioned above is that real-time programming is the execution time constraints; this will, according to Wirth’s terminology, be the difficulty involved in the interleaved access to shared data is eliminated by augmenting parallel programming with monitor like constructs, which can be used to enforce mutually exclusive access to the shared data. The use of synchronization primitives allows the processes order their activities.

As an example: controlling a liquid level in a tank, the case where the liquid reach a high level (alarm), the physical process does not wait for the control program to read the sensor and invoke an emergency action to the process. Some point in the program must ensure that certain conditions are satisfied before the event occurs.

Timing constraints in real-time programming do depend on timing constants of the physical process. Taking the precedent example, there may be some reasoning about tank overflow duration as time constant system dependant.

II. THE APPROACH

Let give a brief summary of some proposed approaches in using temporal logic (or other model) to reason about time in programming. We intend to classify them into three categories:

i- Reasoning about time at the instruction level

ii- Reasoning at the specification level

iii- Reasoning at the model level

This bottom-up classification enables to see how the timing properties can tackled in complementary steps. Our concern in the present work will be focused on (ii) and (iii).

Reasoning about time at instruction level: Timing properties such deadlines, periodic execution of processes, and external event recognition are studied at the instruction level. The main contributions are due to Haase [5] and Shaw [6]. Such scheme has been extended to concurrent programming. From this, two main ideas have emerged; the first that upper and lower bounds on execution time for instruction can be derived, based on given bounds for primitive statements in the language. The second is the use of Dijkstra guarded commands (or Hoare logic for Haase approach) to include the effects of updating real-time. There was an attempt of idealisation of real-time as realized by a perfect global clock denoted by rt, the computer time by ct and the relation between the two expressed as

$$ct = rt + \delta, |\delta| \leq \epsilon, \delta \text{being the clock drift.}$$

The major outstanding issue is whether or not useful best or worst case execution time bounds can be foundfor statements in contemporary high level languages. The main contribution of this type of approach is the technique that, inprinciple, permits the prediction of the timing as well as the logic behavior of programs.

Dealing with time at the specification level: Most approaches for specifying real-time systems (or in more general program behaviour) combine transition systems models (state machines (SM), extended SM (ESM), Petri net (PN)) for modelling a system (software or hardware components) and temporal logic for specifying system behaviour in terms of properties (timing, fairness, termination,etc).

Limitation: Although the approach seems interesting for discreteevent system (DES); we may overcome this by considering continuous systems (CS) and DES under a generalized mode by emphasizing more on time occurrences of stimulus.

Dealing with time through the model approach: The approach is similar to the above if we associate a model to a language (specification language). However, in order to avoid any confusion let give some terminological definition. We see by a model a defined entity of a system expressed by some theories (functional analysis, logic, etc.). This definition is control system oriented. The main idea is by taking into account the critical nature of real-time system application and the extension of previous work on concurrent programs models [7,8,9,10]. The model should express the timing aspect: that will be the main extension. An execution model is developed and an associated proof system has been obtained in [11].

There is an attempt in the present work to define a computation model. The originality of this model is that it involves both *dynamics* of the components contained in the reactive system: the computer process and the physical process. The model consists of communicating processes; the communication model being CSP. Each process, computer process or physical process is modelled as a state transition system.

III. TEMPORAL LOGIC

The version of temporal logic we will use was developed by PNUELI [12] and further described by LAMPORT [7]. A discussion of its use in program verification can be found in many papers of the authors cited above. The two main applications of temporal logic are artificial intelligence and software engineering. The first in on a proper treatment of time, since an action takes place in time and requires an appreciation of the logic structure of temporal facts. The main temporal logics are the one based on first order logic [McDERMOTT-82] and Allen's theory of time [ALLEN-83]. In software engineering, many steps are required, among them specification, synthesis and validation (testing and verification). The temporal logic of programs introduced by Pnueli [PNUELY-77] has been fundamental on the field for the last decade.

In order to get a flavor, let take an example. Having the following algorithm written in a structured pascal like program:

```
y1:=n;
y2:=m;
while y2 > 0
        do
                y1:=y+1;
                y2:=y2-1;
        od.
```

Which calculates two numbers $y1$and $y2$. Such sequential program when executed is an active program, called computer proves in our terminology. This can be represented by a state diagram shown in figure 1.

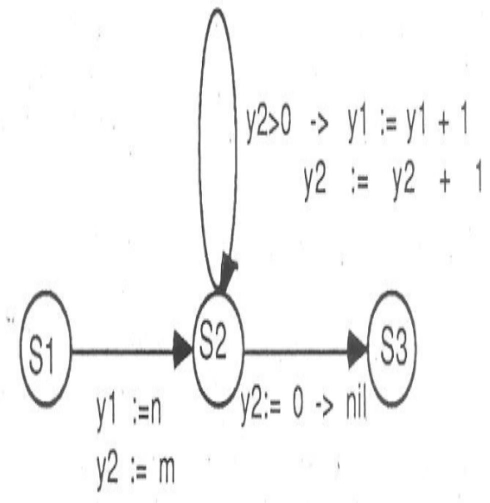


FIGURE 1: State diagram of a computer process

There is a condition and an assignment. The states denoted by Si can be linked through the modal formula:

$$S1: cond(S1, S2)Y ---\rightarrow Y := f(\overline{S1}, S2)(y): S2$$

$y: (y1, y2, \dots, yn)$ represent the variables of the programs
Temporal structure
If we define computational sequence as S0 ----> S1 ----> …

Where Si is a state whose structure is
$S_k =< I, C >$and$I \in \{Ii; i = 1, \dots, n\}$
$C: c1, c2, \dots, cn$; values taken at state Si by variables Y.
A real-time system consists of several processes:
- Computer processes
- Physical processes

The possible states can be contained into a set $\sum = \{s0, s1, s2, \dots\}$ and a reachability relation defined by $\{s_i, s_j\}$ such that $i \leq j$.

The temporal specification language: The common temporal operators added on the ones used in propositional logic are □ and ◊and defined as:
□A is true in state s_i if $\forall s_j$ such that $(s_i, s_j) \in R$ and A is true in s_j.
◊A is true in state s_i if $\exists s_j$ such that $(s_i, s_j) \in R$ and A is true in s_j.
Programming with temporal formula:
If $V = \{v1, \dots, vn\}$ n variables different from those appearing in the program; the associated formula will be:

$$((at\ I_1 \wedge cond((I_1 I_2)(Y) \wedge Y = V) \rightarrow \Diamond(at\ I_2 \wedge Y = f(I_1 I_2))$$

This means that at each instant during execution; if the control is in I_1 and $cond(I_1 I_2)$ is verified, then at that instant or later the control will be in I_2 and the assignment $Y = f(V)$ will be executed.
Concerning the above program, we will apply this to it:

a- at $I_1 \wedge y_1 = v_1 \wedge y_2 = v_2 \rightarrow \Diamond(at\ I_2 \wedge y_1 = n \wedge y_2 = m)$

Such formula indicates that if control is at initial state (I_1) then there exists a reachable state where $y_1 = n$ and $y_2 = m$ and the control will be at I_2.

b- $((at\ I_1 \wedge y_2 > \quad 0\ y_1 = v_1 \wedge y_2 = v_2) \rightarrow \Diamond(at\ I_n \wedge y_1 = v_1 - 1 \wedge y_2 = v_2 - 1)$

Formula in (b) states that at any instant during execution, if control is at initial state (I_1) and condition $y_2 > 0$ is checked, there exists a future state where $y_1 = v_1 - 1$ and $y_2 = v_2 - 1$ have been executed and control is at I_n.
c- Lastly for the other transition $(I_2 \rightarrow I_3)$ we will have the following formula:

$$((at\ I_1 \wedge y_2 = 0 \quad y_1 = v_1 \wedge y_2 = v_2) \rightarrow \Diamond(at\ I_3 \wedge y_1 = v_1 \wedge y_2 = v_2))$$

Some invariant properties can be checked in the following way:
$P1 \rightarrow P2$ If P1 is true then P2 will be true now and in the future
$P1 \rightarrow \Diamond P2$ If P1 is true then P2 will eventually be true in the future (there exists a future where P will be true).

Partial correctness: It is specified by $(at\ I_1 \wedge P) \rightarrow (at\ I_n \wedge Q)$

P is at initial state and if the program terminates then Q must be true.

Total correctness: It is specified by $(at\ I_1 \wedge P) \rightarrow \Diamond(at\ I_n \wedge Q)$

P is true at the initial state, then termination is guaranteed and Q is verified. Going back to the same program example; the partial correctness is expressed as

$$at\ I_1 \rightarrow (at\ I_3 \rightarrow y_2 = n + m)$$

If terminal state is reached then $y = n + m$

And the total correctness as: $at\ I_1 \rightarrow \Diamond(at\ I_3 \wedge y_2 = n + m)$ (The terminal state will be reached with $y = n + m$)

IV. THE COMPUTATION MODEL

The model is defined as <PP,CP,CC> where

$PP = \{pp_1, pp_2, \dots, pp_n\}$, set of physical processes

$CP = \{cp_1, cp_2, \dots, cp_m\}$, set of computer processes

$CC = \{cc_1, cc_2, \dots, cc_k\}$, set of communicating channels

Transition system plays an important role for describing and analyzing processes and systems of communicating processes. We make use of it for state description of processes. The TMRS, transition model for real-time systems, differs from common model encountered in the literature in one particular aspect: the insertion of physical processes in the model.

Generally a process is a set of states and that an action or an event makes the current state of the process to change. Thus the possible behaviours of a process are represented by transitions; each transition contains the current state of the process, the new state it enters and the name of action or event which caused the change. An application on a sequential program has been presented in the previous section.

Let consider the following control program:

```
Program control is
        X,y: address;
        S:sensor_value;
        A:actuator value
        While true do
                Read_port(s,x);
                Calculate error;
                Calculate control;
                Write_port(s,y);
        Od.
```

This is the simplest form of a control program; there can be two possible enhancements to it; the first is to set a timer to each loop by using a timing primitive at the while statement, such enhancement is synchronous timing; another possible construct is to set predicate the state change at the s port; very often we make use of the first in normal execution and of second in case of critical action to be invoked by using interrupt constructs. The main assumption is that the time response of the program is less than the one of the system being controlled: this main idea is exploited in the development of specific control languages (synchronous languages).

In real-time systems, program interacts continuously with the external environment. An important issue is how to take real-time constraints into considerations. It is shown [2] that we cannot rely on timing information obtained from the static program text since we assume that the processor may be shared among the various processes; obviously, we need to know something about the strategy by which it serves individual processes and, in particular, whether or not certain processes are served with priority.

Our approach is as follows:

- Defining the computation model
- Specifying timing properties using temporal logic
- Develop a model checker to get diagnosis about timing properties of the system.

The present work is limited to the two first steps. The model checking may be carried by using an extended version of existing ones (such as EMC, MEC, XESAR).

A fundamental assumption is the controllability/observability (for continuous systems) of the physical process. For discret event systems, the assumption is implicitly done in the specification step when using Petri nets for example. Such properties can be handled using state space theory for continuous systems and Petri nets theory for discret event systems. Almost all previous work were concerned with computer processes only in their models. The physical process is being considered for two reasons:

i. A global computation model for real-time systems cannot be completely expressive without one main component as the external environment; this is a common argument.

ii. There are many information to be gained when considered the external environment. The time constants corresponding to the dynamic modes are as fundamental as the time execution of a program we saw in dealing with time at the instruction level.

iii. Processes interact through messages; such interaction is formalized using CSP constructs (message passing with null size buffer). In this type of communication, two processes must be both at their respective synchronization points. For two communicating computer processes we have: $cp_i!\,cc_1$ and $cp_j?\,cc_1$ this means that processes i (sender), j (receiver) communicate on channel 1. We can also communication between a physical process and computer process expressed as $cp_i!\,cc_2$and $cp_j?\,cc_2$.

The model of real-time message passing is a standard one [16]:

M: $TIME \rightarrow COM \times WTS \times WTR$ (COM: Communicating channels, WTS: Wait to send, WTR: Wait to receive)

M .com: set of communicating channels M .wtr: set of channels waiting to receive

M .wts: set of channels waiting to send

The critical point is when a pp_i is waiting to send or waiting to receive. These two cases have to be avoided. In order to get around the problem, we categorise the timing interaction into four cases, approach proposed in [17]. The timing parameters, denoted as $t_{pp}, t_{cp}, t_{pc}, t_{cc}$ are maximum time allowed if the four following cases:

t_{pp}: two stimulus (from pp to cp in two successive steps): Data acquisition process

t_{cp}: Between an action and corresponding stimulus (From cp to pp and pp to cp)

t_{pc}: between a stimulus and corresponding action (from pp to cp and cp to pp)

t_{cc}: Between two actions (from p to pp in two successive steps): $cp_i!\,cc_1$ and $cp_j?\,cc_1$

It appears that the most important parameter related directly to the timing constraints (or has more weight on it) is: t_{pc}

The physical process: In order to illustrate the mapping of the physical system into a process, we make the assumption that the system is linear and hence can be described using state space theory as

$$X = A.X + B.U \text{and} Y = C.X$$

U being the control vector, X state vector.

A, B, C matrices corresponding to the physical process parameters

The matrix A defines the dynamics of the system (eigenvalues of A correspond the time constants inverses of system dynamic modes). A supplementary and important assumption is that allsystem modes are observable and controllable. The system can be represented by a state transition system. The state being the X vector; we consider this statement with great attention because of the state explosion problem; we may consider some finite intervals as to limit the number of possible values that can be taken of the elements of X. It was shown that both pp_i and cp_i can be modelled using a state system transition. The interaction is modelled using real-time message passing contruct.

V. SPECIFYING TIMING PROPERTIES

Timing constraints: Let now give some basic definitions:

Definition 1: Computer process sends message through channels on a CSP based mechanism and a sensor synchronization is communication event between cp_i as a receiver and pp_j as a sender.

Definition 2: An actuator (action) synchronization is a communication event between cp_i as sender and pp_j as a receiver.
Definition 3: A send/respond time constraint from precedent definition as the time imposed on the duration between sensor synchronization and actuator synchronization such that the actuator synchronization is triggered following the sensor synchronization.

This means that the sensor synchronization is an event causing a transition on the cp state, hence delivering an action causing an actuator synchronization.

Timing properties and timing specification: Our general interest is the real-time programs has the right properties. By decomposing each process into states, we make use of the available results to check:

- Fairness
- Deadlock freeness
- Timing properties

For similar approaches, see [18,19]; for a survey on general approaches see [20].

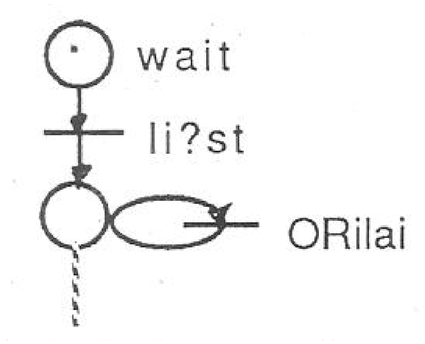
Case study: Let take a physical process consisting of two robots for manipulating manufactured parts and two spaces, one where the parts are stored, the other where the parts to be put for assembly.

Let define a set of activities $A = \{waite, pick, put, move\}$

$a_i$being the variable, current state of robot i

Data variable p_i, the number of parts in space i

Communicating channels: output channels ORi and OSi to send ai and di to the control process control (respectively from robot and space). Input channels Ii to receive st and as from the control process to enter the store or the assembly space. Using CSP, the communication semantics can be described as: **ORi!ai** means robot I is sending its state ai on channel ORi and in the same way**Ii?st** means the robot is receiving an order st on channel Ii (to enter the store space). The dynamics of the physical process can described using Petri nets, let give a partial description as follows:



The control process can described as set of sequence of executions, each sequence is itself a sequence of states and events $\Sigma = s_0 s_1 s_2 \ldots (s_k, k^{th}\ sequence\ in\ \Sigma)$.

If we use temporal logic formula, ie if w is a state formula, we denote by Sat w, that is the formula is satisfied in s_0. Having w_1, w_2 temporal formula we can express some temporal properties $w_1 and\ t = T \Rightarrow \Diamond(w_2 and\ t \leq T + n)$; If w_1 is true now and the time is T and in n time-units (clock pulses), the formula w_2 must become true. Such temporal formula will be used to make sure that some timing constraints are met.

Another interesting aspect in temporal logic is to set a safety formula for mutual exclusion in the case study:
$(a_1 \in \{move, wait\} or\ x_2 \in \{move, wait\})$; this invariant property must be always true in order to avoid collision.

CONCLUSION

It was presented an approach to model real-time systems and how to dal with its properties. The main contribution is the taking into account both types of processes and that physical processes can be modelled using transition systems, a common model for computer and physical processes. Reasoning about time using temporal logic was outlined; an extension of logic to use physical time is possible.

REFERENCES

1. L. Lamport: A simple approach to specifying concurrent programs. DEC research center report, No. 15, Dec 86, revised Jan 88.
2. N. Wirth: Towards a discipline of real-time programming. Comm. ACM, August 77.
3. S. Young: Real-time languages, E. Howood, 1982.
4. H. H. Rosenbrock: State space an multivariable theory. Prentice Hall, 1976.
5. V. Haase: Real-time behavious of programs. IEEE Trans. Soft. Eng, Vol. SE-7, No. 5, Sept 81.
6. A.C. Shaw: Reasoning about time in higher-level language software. MASI rep No. 188, Univ Paris VI, July 87.
7. L. Lamport: Specifying concurrent program modules. ACM, TOPLAS, Vol. 4, No. 3, July 87.
8. S. Owicki, L. Lamport: Proving properties of concurrent programs. ZCM, TOPLAS, Vol. 4, No. 3, July 82.
9. A. Pnueli: Specification and development of reactive systems. Information Processing, Proc. IFIP, 1986.

10. Z. Manna, A. Pnueli: How to cook a proof system for your pet language. Proc. Symp. On POPL, Jan 83.
11. A. Bernstein, P. Harter: Proving real-time properties of programs with temporal logic. Proc. 8thsymp.Operating systems principles, ACM SIGOPS, 1981.
12. A. Pnueli: The temporal logic of programs. Proc. 18th symposium foundations of comp science, Oct 77.
13. D. McDeermott: A temporal logic for reasoning about processes and plans. Cog. Sci, No.6, 1982.
14. J. Allen: Towards a general theory of action and time. Artificial intelligence, No. 23, 1984.
15. L. Farinas et al: Théorie de la programmation et logique temporelle. TSI, Vol 6, 1987.
16. J. Hooman, J. Widom : A proof system for message passing. Tech. report, No. 88-919, Cornell Un, June 88.
17. Dasarathy: Timing constraints of real-time systems. IEEE trans. Soft-Eng, Vol. SE-11, No.1, Jan 85.
18. F. Jahaniam, A. K. Mok: Safety analysis of real-time systems. IEEE Trans. Soft-Eng, Vol. SE-12, No.9, Sept 86.
19. J. OStroff: Real-time computer control of discret event systems modelled by extended state machines: a temporal logic approach. Tech. Report 8618. Systems control group, Univ. Toronto, Jan 87.
20. A. Sahraoui: Timing constraints problems in real-time systems: A survey. LAAS-CNRS Tech. Report, No. 88346, Toulouse, October 88.